CompSci 316 Fall 2017: Homework #4

100 points (8.75% of course grade) + 10 points extra credit
Assigned: Tuesday, November 14
Due: Tuesday, December 5

This homework should be done in parts as soon as relevant topics are covered in lectures. If you wait until the last minute, you might be overwhelmed.

For Problems 1 and 4, you will need to use Gradiance. Access Gradiance via the "Gradiance" link on the course website. There is no need to turn in anything else for these problems; your scores will be tracked automatically.

For other problems, you will need to turn in the required files electronically. Please read the "Help → Submitting Non-Gradiance Work" section of the course website for instructions. When submitting your work, make sure you select the correct course and homework. Multiple submissions are okay, but please upload ***all*** required files in each resubmission.

For Problems 2, 3, and X1, you may prepare your answers electronically or on paper (handwritten). In the latter case, scan or photograph the pages and submit the resulting PDF (preferred) or JPG files. Please name your files informatively, e.g., as n`.pdf`, where n is the problem number.

Problem 1 (15 points)

Complete the Gradiance homework titled "Homework 4.1 (Indexes)."

Important note: These problems use a definition of "fan-out" that is different from what we discussed in class. When they say "fan-out $n = 3$" they mean that the maximum number of keys per node is 3, and the maximum number of pointers per node is 4 (i.e., "max fan-out is 4" in our terminology).

Problem 2 (20 points)

A table $R(\underline{K}, A, \ldots)$ with $100{,}000$ rows is stored in $10{,}000$ disk blocks. The rows are sorted by R's primary key K, but not by A. There is a dense, secondary B$^+$-tree index on $R(A)$, which has 3 levels and 500 leaves.

Suppose we want to sort R by A. We have 101 memory blocks at our disposal. Method 1 performs an external-memory merge sort using all memory available. Method 2 takes advantage of the fact that the values of A are already sorted in the B$^+$-tree index on $R(A)$: It simply scans the leaves of the index to retrieve and output R rows in order.

How many disk I/O's do these two methods require? Which one is the winner?

Problem 3 (45 points)

Consider the following schema for an online bookstore:

Cust (*CustID*, *Name*, *Address*, *State*, *Zip*)
Book (*BookID*, *Title*, *Author*, *Price*, *Category*)
Order (*OrderID*, *CustID*, *BookID*, *ShipDate*)
Inventory (*BookID*, *Quantity*, *WarehouseID*, *ShelfLocation*)
Warehouse (*WarehouseID*, *State*)

Cust and *Book* represent customers and books, respectively. When a customer buys a book, a tuple is entered into *Order*. *Inventory* records the quantity and shelf location of each book for every warehouse. *Warehouse* records the state where each warehouse is located in. *Price* is numeric. *ShipDate* is an integer representation of a date. In the following, `:today` is a constant denoting the integer representation of today's date.

(a) Transform the following query into an equivalent query that 1) contains no cross products, and 2) performs projections and selections as early as possible. Represent your result as a relational algebra expression tree.

$\pi_{Title,Author}$
$\sigma_{(Zip=27708)\text{ and }(Title\ LIKE\ \text{"\%}Steve\ Jobs\text{\%"})\text{ and }(Price<20)}$
$\sigma_{(Cust.CustID=Order.CustID)\text{ and }(Book.BookID=Order.BookID)}$
$(Cust \times Order \times Book)$.

Suppose we have the following statistics:

- $|Cust| = 1{,}000;\ |\pi_{State}Cust| = 50;$
- $|Book| = 600; |\pi_{Category}Book| = 10;$
- $|Order| = 60{,}000; |\pi_{BookID}Order| = 600; |\pi_{CustID}Order| = 1{,}000; |\pi_{ShipDate}Order| = 2{,}000;$
- $|Inventory| = 30{,}000; |\pi_{BookID}Inventory| = 600; |\pi_{WarehouseID}Inventory| = 50;$
- $|Warehouse| = 50; |\pi_{State}Warehouse| = 50.$

For each of the following relational algebra expressions, estimate the number of tuples it produces. Note that each estimation may build on the previous ones. You may make the same assumptions as in the lecture on query optimization. You may also make different or additional assumptions, but please state them explicitly.

(b) $\sigma_{ShipDate>:today-60}Order$.
(c) $\pi_{CustID}(\sigma_{ShipDate>:today-60}Order)$.
(d) $\sigma_{State="NC"}Customer$.
(e) $(\sigma_{State="NC"}Customer) \bowtie (\sigma_{ShipDate>:today-60}Order)$.

For the following questions, further suppose that:

- Each disk/memory block can hold up to 10 rows (from any table);
- All tables are stored compactly on disk (10 rows per block);
- 5 memory blocks are available for query processing.

(f) Suppose that there are no indexes available at all, and records are stored in no particular order. What is the best execution plan (in terms of number of I/O's performed) you can come up with for the query $\sigma_{ShipDate=:today}(Order \bowtie Inventory)$? Describe your plan and show the calculation of its I/O cost.

(g) Suppose there is a B+-tree primary index on *Order*(*OrderID*) and a B+-tree primary index on *Inventory*(*BookID*, *WarehouseID*), but no other indexes are available. Furthermore, assume that both B+-trees have a maximum fan-out of 100 for non-leaf nodes; each leaf stores 10 rows; and all nodes in both B+-trees are at maximum capacity except the two roots. What is the best plan for the same query in (f)? Again, describe your plan and show the calculation of its I/O cost.

Problem 4 (20 points)

Complete the Gradiance homework titled "Homework 4.4 (Concurrency Control and Recovery)."

Extra Credit Problem X1 (10 points)

Consider a table R with 100,000 rows, which are stored compactly on disk in exactly 10,000 blocks in no particular order. There are only 20,000 distinct rows in R (other rows are duplicates of these). We wish to compute the following query Q: `SELECT DISTINCT * FROM R;`

(a) What is the minimum amount of memory (in blocks) required to compute Q in one pass (i.e., using only 10,000 I/O's excluding the cost of writing the result)?

(b) Suppose that we only have 1,001 blocks of memory available. Devise a strategy that can compute Q in no more than 20,000 I/O's in the worst case (again, excluding the cost of writing the result).